Learning Python

Getting results for beamlines and scientific programming

Using python: Plotting with matplotlib in wxPython apps

Outline of topics to be covered

Using the plotnotebook.py class to combine wxPython and matplotlib

1. Multiple tabs in one window
2. Multiple plot windows
3. Adding more graphics to a plot
4. Redrawing the contents of a plot
5. Adding a widget to a plot
6. Responding to a key press
7. Showing the mouse position

Using matplotlib with wxPython

- We saw earlier how to use matplotlib, but life changes when we want to use wxPython and matplotlib together. Why? In part because matplotlib uses wxPython for display. pyplot and wxPython will both try to control the event loop unless we use a bit more care.

- To make life easier, I provide a short module, plotnotebook, that does much of the setup needed to use the two packages together. It can be used directly for programs that will only use simple wxPython widgets on the plot frames

  – Basic use:

    import plotnotebook

    Make window(s) with `MakePlotWindow`()

    Make plot tabs with `AddPlotTab`

    Do graphics on tab with matplotlib

    Call `ShowPlots` to start event loop

```
import plotnotebook
win = plotnotebook.MakePlotWindow()
tab = win.AddPlotTab('Plot1')
ax = tab.figure.gca()
ax.plot(range(10),range(10),’o-')
plotnotebook.ShowPlots()
```

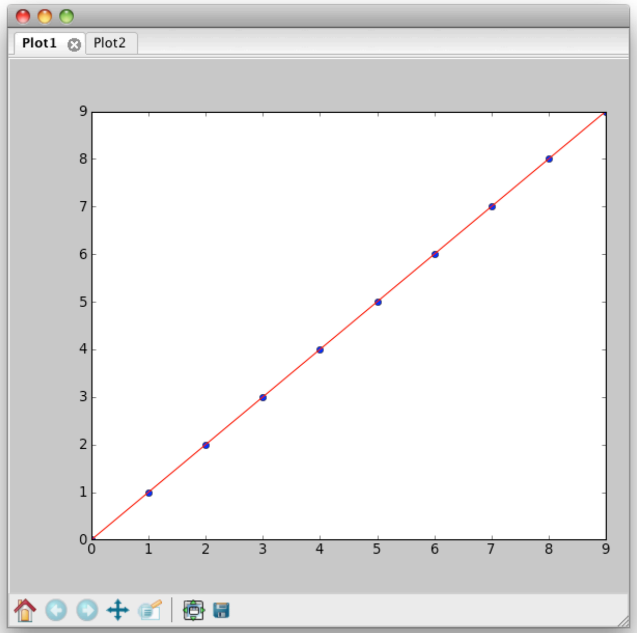
pnb_demo1.py

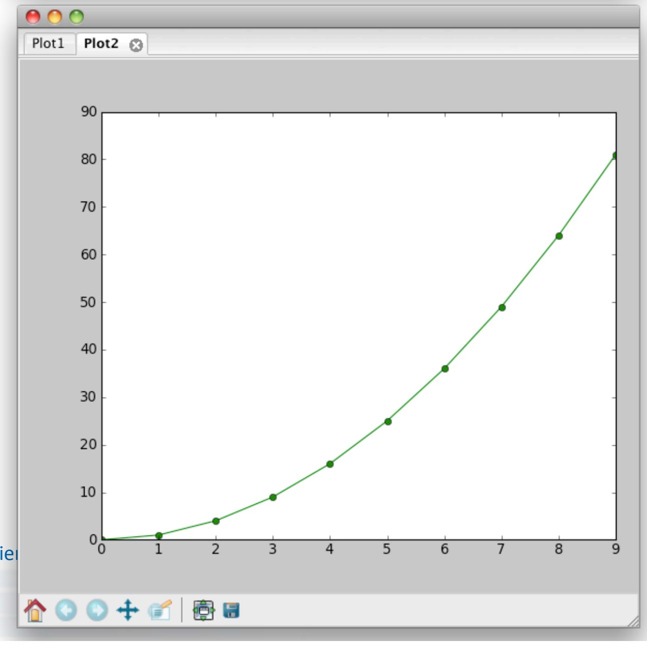
Plotting with multiple tabs in a window

- Call AddPlotTab multiple times to add additional plots into a single window

```
import plotnotebook
win = plotnotebook.MakePlotWindow()
tab1 = win.AddPlotTab('Plot1')
ax1 = tab1.figure.gca()
ax1.plot(range(10),range(10),'o')
ax1.plot(range(10),range(10),'-r')
tab2 = win.AddPlotTab('Plot2')
ax2 = tab2.figure.gca()
x2 = [i**2 for i in range(10)]
ax2.plot(range(10),x2,'o-g')
plotnotebook.ShowPlots()
```

pnb_demo2.py



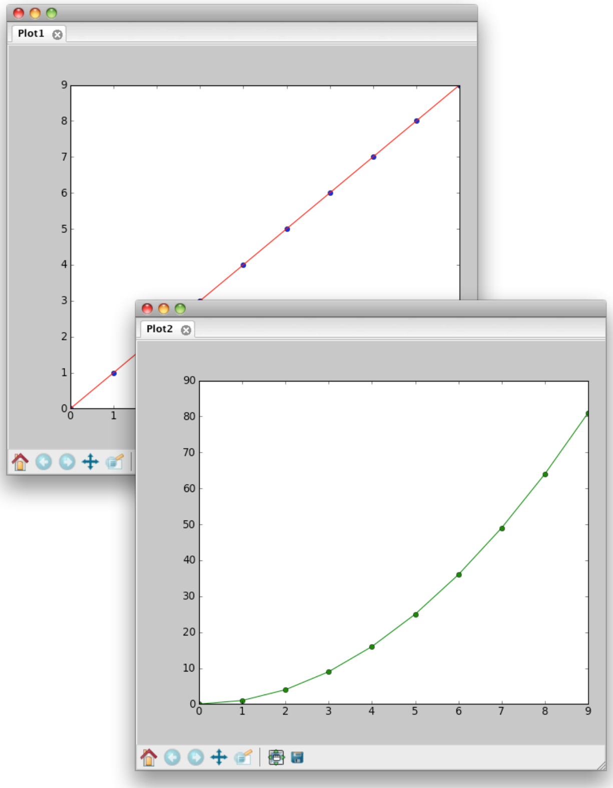


Plotting with multiple windows

- Call `MakePlotWindow` multiple times to create multiple plot windows

```
import plotnotebook
win1 = plotnotebook.MakePlotWindow()
tab1 = win1.AddPlotTab('Plot1')
ax1 = tab1.figure.gca()
ax1.plot(range(10),range(10),'o')
ax1.plot(range(10),range(10),'-r')
win2 = plotnotebook.MakePlotWindow()
tab2 = win2.AddPlotTab('Plot2')
ax2 = tab2.figure.gca()
x2 = [i**2 for i in range(10)]
ax2.plot(range(10),x2,'o-g')
plotnotebook.ShowPlots()
```

pnb_demo3.py



  – Of course one can create as many tabs as desired inside a plot window

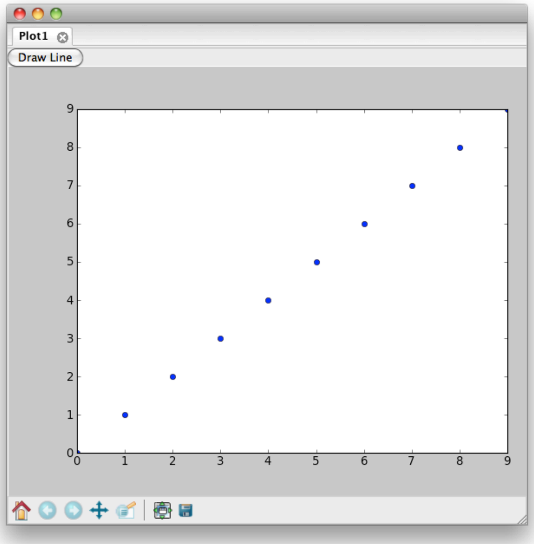
Updating the contents of a plot

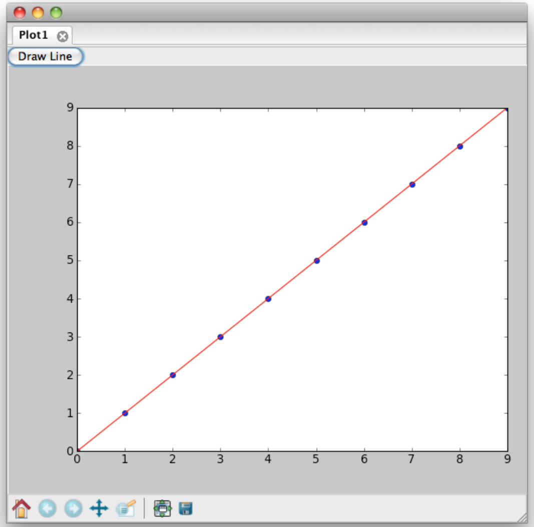
- To add more items to the plot, one should make the appropriate calls, such as plot and then call <plot>.canvas.draw()

```
import plotnotebook
import wx
win1 = plotnotebook.MakePlotWindow()
tab1 = win1.AddPlotTab('Plot1')
ax1 = tab1.figure.gca()
ax1.plot(range(10),range(10),'o')
def AddLine(event):
    tab = event.GetEventObject().GetParent()
     x = range(10)
    tab.figure.gca().plot(x,x,'-r')
    tab.canvas.draw()
btn = wx.Button(tab1, -1, label='Draw Line')
btn.Bind(wx.EVT_BUTTON, AddLine)
tab1.topsizer.Add(btn, 0, wx.LEFT)

plotnotebook.ShowPlots()
```

pnb_demo4.py





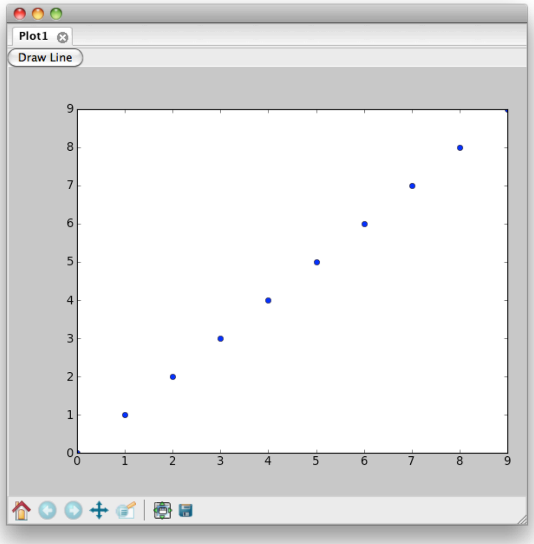
Redrawing an existing plot

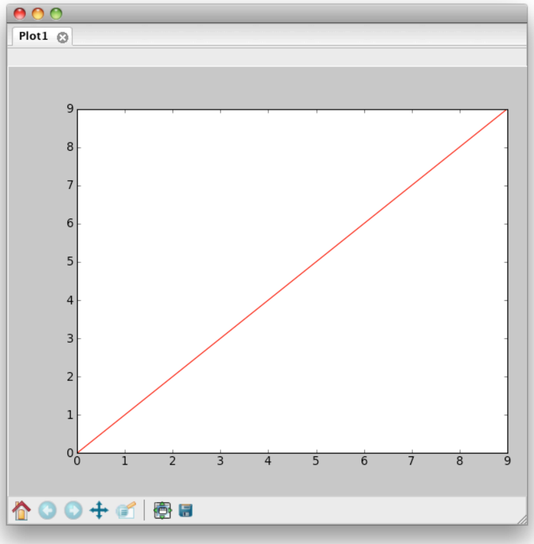
To make a new plot, first use <plot>.`figure.clear()` to delete the old plot

- Note that here **`tab.figure.gca()`** creates axes
- Also, **`tab.topsizer.DeleteWindows()`** deletes the contents of topsizer

```
import plotnotebook
import wx
win1 = plotnotebook.MakePlotWindow()
tab1 = win1.AddPlotTab('Plot1')
ax1 = tab1.figure.gca()
ax1.plot(range(10),range(10),'o')
def AddLine(event):
    tab = event.GetEventObject().GetParent()
    tab.figure.clear()
     x = range(10)
    tab.figure.gca().plot(x,x,'-r')
    tab.canvas.draw()
    tab.topsizer.DeleteWindows() # del button
btn = wx.Button(tab1, -1, label='Draw Line')
btn.Bind(wx.EVT_BUTTON, AddLine)
tab1.topsizer.Add(btn, 0, wx.LEFT)
plotnotebook.ShowPlots()
```

pnb_demo5.py





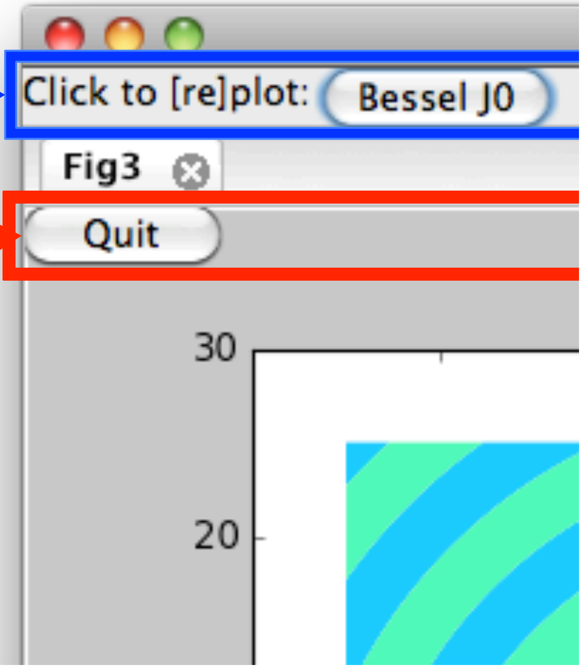
Adding a button to a plot

- There are two sizers provided for placing wxPython widgets (typically buttons)
  - The one at the top is associated with the window (frame)

```
win1 = plotnotebook.MakePlotWindow()
lbl = wx.StaticText(win1, -1, "Click to [re]plot: ")
win1.topsizer.Add(lbl, 0, wx.LEFT)
btn = wx.Button(win1, -1, label='Bessel J0')
btn.Bind(wx.EVT_BUTTON, SetJ0)
win1.topsizer.Add(btn, 0, wx.LEFT)
```

  - The one just below is associated with the tab, and will change with the tab

```
page = win1.AddPlotTab('Fig3')
btn = wx.Button(page, -1, label='Quit')
btn.Bind(wx.EVT_BUTTON, sys.exit)
page.topsizer.Add(btn, 0, wx.LEFT)
```



pnb_demo6.py

Responding to a key press in the window

- matplotlib provides a mechanism to connect bind a function that will be called when any key press is pressed inside a plot:

```
page1 = plotter.AddPlotTab('Plot1')
page1.figure.canvas.mpl_connect('key_press_event', onKeyPress)
```

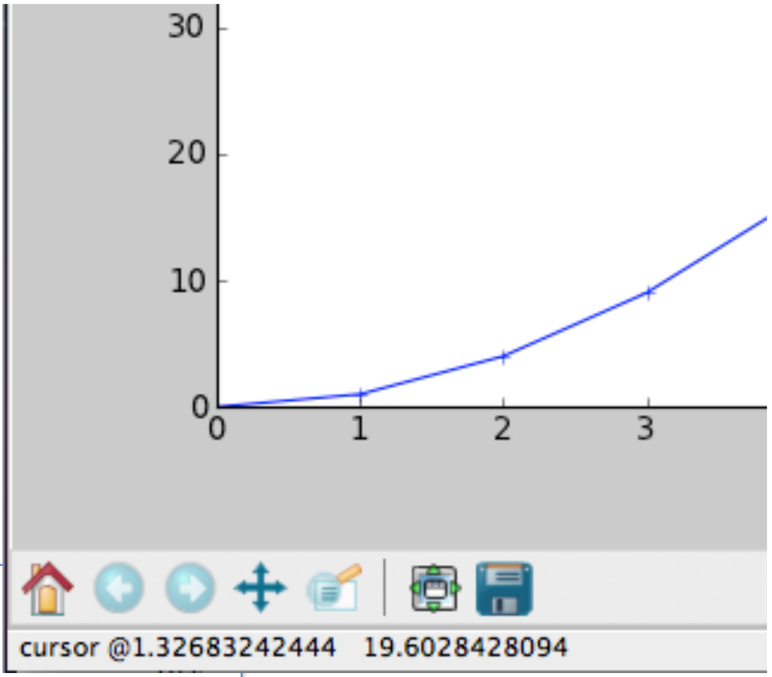
- The referenced routine will be called when any key is pressed (including shift, control,…). The routine will need to figure out what it can ignore from the event object

pnb_demo7.py

```
def onKeyPress(event):
    "sample routine to respond to key press"
    key = event.key
    if len(key) > 1: return # ignore shift, etc.
    if event.guiEvent.ShiftDown(): key = key.upper()
    if event.xdata is None or event.ydata is None:
        print "Key",key,"was pressed outside the plot"
    else:
        print "Key",key,"was pressed @",event.xdata, event.ydata
```

Showing mouse position on status bar

- We add a status bar to the frame
- Stick a reference to the status bar where we can get it in the event handler
- The event handler creates a character string and puts it in the status bar



cursor @1.32683242444   19.6028428094

```
def OnMotion(event):
  x,y = event.xdata,event.ydata
  statbar = event.canvas.statbar
  if x is not None and y is not None:
    statbar.SetStatusText('cursor @'+str(x)+'   '+str(y))
  else:
     statbar.SetStatusText('cursor outside window')
```

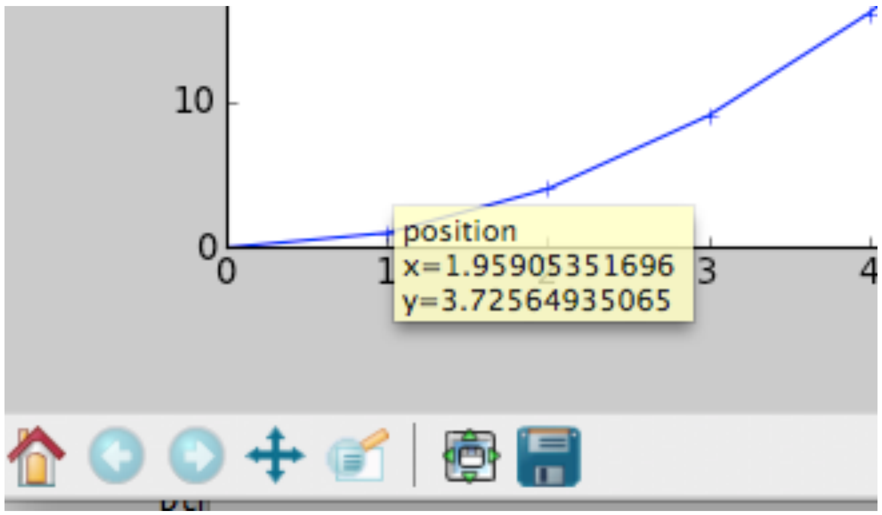
pnb_demo8.py

```
# add a status bar to the frame & save reference in objects
statbar = plotter.frame.CreateStatusBar()
page1.figure.canvas.statbar = statbar
page2.figure.canvas.statbar = statbar
# code to respond to mouse movement
page1.figure.canvas.mpl_connect('motion_notify_event', OnMotion)
page2.figure.canvas.mpl_connect('motion_notify_event', OnMotion)
```

Showing mouse position with tooltop

A tooltip pops up when the mouse is left stationary in a spot for a little while.

The tooltip is being created every time the mouse is moved, but is posted only if the mouse stops moving



position
x=1.95905351696
y=3.72564935065

```
def OnMotion(event):
  x,y = event.xdata,event.ydata
  if x is not None and y is not None:
    event.canvas.SetToolTipString(
        'position\nx='+str(x)+'\ny='+str(y))
  else:
      event.canvas.SetToolTipString('')
```

pnb_demo9.py

```
# code to respond to mouse movement
page1.figure.canvas.mpl_connect('motion_notify_event', OnMotion)
page2.figure.canvas.mpl_connect('motion_notify_event', OnMotion)
```